PeekabooEvent

Modules

math operator types

Classes

PeekabooEvent

class **PeekabooEvent**

```
Represents an event in a PeekabooRound. Should not be initialized
by the user.
```

Methods defined here:

__init__(self, data)

```
    takes a data list produced by PeekabooRound
```

PeekaboomRound

Modules

copy math operator types

Classes

PeekaboomRound

class **PeekaboomRound**

```
Represents one round of play: an exchange between a peek player and
a boom player concerning a single object. A PeekaboomRound is
initialized with one line of data from a .pkb file. The initializer
parses the line and assigns properties to the PeekaboomRound
instance accordingly.

 Events are stored as PeekaboomEvents, and blob events can be
accessed independently with the blobs() method. The Point2 objects
(from the euclid module) associated with those blob events can be
accessed with blobPoints(), and a PointCluster object can be
generated from those points with the cluster() method.

 The merge() method is used to non-destructively join multiple
PeekaboomRound objects into a single object. The resulting object
ends up representing the events from multiple rounds of play and is
useful for initializing a Photographer object with all the data for
a particular label in an image.
```

Methods defined here:

__init__(self, data)
```
takes a single line from a .pkb file
```

blobPoints(self)
```
returns a list of points from all blob events
```

blobs(self)
```
returns a list of blob events
```

cluster(self)
```
returns a PointCluster of blobPoints()
```

merge(self, others)
```
takes other PeekaboomRounds and returns a new PeekaboomRound with
properties of this one and the events from all
```

Photographer

index
../Photographer.py

Modules

copy   math   operator   types

Classes

Photographer

class **Photographer**

```
The Photographer class is initialized with a PeekaboomRound object
and is responsible for evaluating a single object's adherence to
the rule of thirds. This returned through the Photographer's
ruleOfThirds() method as a value between 0.0 and 1.0.

 The Photographer considers the image to be squashed into a square;
that is, (x,y) co-ordinates are always expressed by the
Photographer as values between 0.0 and 1.0, where 1.0 is equal to
the width or height of the image respectively. This is to ensure
that proximities are evaluated relative to the image as a whole and
that the focus is on the vertical and horizontal rhythms rather
than absolute pixel values.

 The ruleOfThirds method ends up taking into account two different
factors and returns a weighted average of them:

 The first factor is based on the clusterIsAlongLine() method,
which checks the PeekaboomRound's cluster of points against one of
the Photographer's gridlines and returns a value closer to 1.0 when
the lengthAxis() of the PointCluster is both close in distance and
similar in rotation to the gridline.

 The second factor is returned by the clusterIsAtPoint() method,
which returns a value closer to 1.0 when the PointCluster is close
in distance to a given intersection.

 The Photographer checks the output of clusterIsAlongLine() for
each of the gridlines, and checks the output of clusterIsAtPoint()
for each of the gridline intersections. Only the maximum of these
two sets of values are considered, since the Photographer only
cares about the gridline and/or intersection of best fit for the
cluster.

 The return value of the PointCluster's elongation() method is then
used as the balance for weighting the two maximums. This means
that, for clusters of points which are more elongated, the
Photographer will care more about the cluster being along a
gridline than about its centre being close to one of the
intersections.

 The Photographer's maxDistance property is the maximum distance
```

```
for which one point can be considered "close to" another. The
maxDistance property has been set to one ninth of the width/height
of the image. Similarly, the maxRotation property is the maximum
rotation for which a line can be considered at all "aligned with"
another line, and it has been set to 30 degrees (pi/6 radians).
```

Methods defined here:

__init__(self, peekaboomRound)

clusterIsAlongLine(self, line)

```
returns a value between 0.0 and 1.0 representing how much the
cluster is located along a particular line
```

clusterIsAtPoint(self, point)

```
returns a value between 0.0 and 1.0 representing how much the
cluster is located at a particular point
```

relativeCluster(self)

```
returns a PointCluster from self.peekaboomRound transformed into
relative co-ordinates between 0.0 and 1.0
```

relativePoint(self, p)

```
takes in a point and returns a new point with co-ordinates between
0.0 and 1.0, relative to the image height and width
```

relativeProximity(self, distance)

```
takes a distance and returns the inverse of its relatived value
between 0 and maxDistance (return value goes up when distance goes
down)
```

relativeRotationalAlignment(self, rotation)

```
takes a rotation in degrees and returns its inverse relativized
value between 0 and maxRotation (return value goes up as rotation
goes down)
```

relativize(self, var, extremeA, extremeB)

```
takes a variable and two extremes, and returns a relative value
between 0.0 and 1.0 depending on where the variable is situated
between the extremes
```

rotationBetweenLines(self, line, segment)

```
docstring for rotationBetweenLines
```

ruleOfThirds(self)

```
returns a confidence value for the object of the PeekaboomRound
adhering to the rule of thirds in the image
```

squishUp(self, v)

```
takes in a number between 0.0 and 1.0 and applies a log function
which bunches the distribution towards 1.0
```

weightedAverage(self, balance, extremeA, extremeB)

```
inverse of relativize()

returns a weighted average between two extremes where balance is
between 0.0 and 1.0
```

Data and other attributes defined here:

gridlines = [Line2(<0.00, 0.33> + u<1.00, 0.00>), Line2(<0.00, 0.67> + u<1.00, 0.00>), Line2(<0.33, 0.00> + u<0.00, 1.00>), Line2(<0.67, 0.00> + u<0.00, 1.00>)]

intersections = [Point2(0.33, 0.33), Point2(0.33, 0.67), Point2(0.67, 0.33), Point2(0.67, 0.67)]

maxDistance = 0.1111111111111111

maxRotation = 0.52359877559829882

PointCluster

Modules

math operator types

Classes

__builtin__.list(__builtin__.object)

PointCluster

class **PointCluster**(__builtin__.list)

```
This is a special list type for Point2 objects from the euclid
module. It is a subclass of the built-in Python list class, with
accessor methods for geometric values useful for the Photographer.
```

Method resolution order:

PointCluster
__builtin__.list
__builtin__.object

Methods defined here:

centre(self)
```
returns this cluster's average/centre point as a new Point2
```

elongation(self)
```
returns between 0.0 and 1.0 proportional to how elongated the
cluster is
```

lengthAxis(self)
```
returns a LineSegment2 representing the most distant pair of points
in this cluster
```

widthAxis(self)
```
returns a LineSegment2 perpendicular to lengthAxis(), spanning
the width of the cluster at a point on the lengthAxis() halfway
between the two furthest points' closest points on the lengthAxis()
```

Data descriptors defined here:

__dict__
```
dictionary for instance variables (if defined)
```

__weakref__
```
list of weak references to the object (if defined)
```

Methods inherited from __builtin__.list:

__add__(...)
```
x.__add__(y) <==> x+y
```

__contains__(...)
```
x.__contains__(y) <==> y in x
```

__delitem__(...)
```
x.__delitem__(y) <==> del x[y]
```

__delslice__(...)
```
x.__delslice__(i, j) <==> del x[i:j]

Use of negative indices is not supported.
```

__eq__(...)
```
x.__eq__(y) <==> x==y
```

__ge__(...)
```
x.__ge__(y) <==> x>=y
```

__getattribute__(...)
```
x.__getattribute__('name') <==> x.name
```

__getitem__(...)
```
x.__getitem__(y) <==> x[y]
```

__getslice__(...)
```
x.__getslice__(i, j) <==> x[i:j]

Use of negative indices is not supported.
```

__gt__(...)
```
x.__gt__(y) <==> x>y
```

__hash__(...)
```
x.__hash__() <==> hash(x)
```

__iadd__(...)
```
x.__iadd__(y) <==> x+=y
```

__imul__(...)
```
x.__imul__(y) <==> x*=y
```

__init__(...)
```
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
```

__iter__(...)
```
x.__iter__() <==> iter(x)
```

__le__(...)
```
x.__le__(y) <==> x<=y
```

__len__(...)

```
x.__len__() <==> len(x)
```

__lt__(...)

```
x.__lt__(y) <==> x<y
```

__mul__(...)

```
x.__mul__(n) <==> x*n
```

__ne__(...)

```
x.__ne__(y) <==> x!=y
```

__repr__(...)

```
x.__repr__() <==> repr(x)
```

__reversed__(...)

```
L.__reversed__() -- return a reverse iterator over the list
```

__rmul__(...)

```
x.__rmul__(n) <==> n*x
```

__setitem__(...)

```
x.__setitem__(i, y) <==> x[i]=y
```

__setslice__(...)

```
x.__setslice__(i, j, y) <==> x[i:j]=y

Use  of negative indices is not supported.
```

append(...)

```
L.append(object) -- append object to end
```

count(...)

```
L.count(value) -> integer -- return number of occurrences of value
```

extend(...)

```
L.extend(iterable) --
 extend list by appending elements from the iterable
```

index(...)

```
L.index(value, [start, [stop]]) -> integer --
 return first index of value
```

insert(...)

```
L.insert(index, object) -- insert object before index
```

pop(...)

```
L.pop([index]) -> item --
 remove and return item at index (default last)
```

remove(...)

```
L.remove(value) -- remove first occurrence of value
```

reverse(...)

```
L.reverse() -- reverse *IN PLACE*
```

sort(...)

```
L.sort(cmp=None, key=None, reverse=False) -- stable sort *IN PLACE*;
cmp(x, y) -> -1, 0, 1
```

Data and other attributes inherited from __builtin__.list:

__new__ = <built-in method __new__ of type object at 0x1ede60>

```
T.__new__(S, ...) -> a new object with type S, a subtype of T
```